QSM368ZP&SC362Z-AP&SG368Z&SH603ZA
Android&Linux RKNN 用户指导

智能产品

版本：1.1

日期：2025-12-08

状态：受控文件

上海移远通信技术股份有限公司（以下简称“移远通信”）始终以为客户提供最及时、最全面的服务为宗旨。如需任何帮助，请随时联系我司上海总部，联系方式如下：

上海移远通信技术股份有限公司
上海市闵行区田林路 1016 号科技绿洲 3 期（B 区）5 号楼　　邮编：200233
电话：+86 21 5108 6236　　邮箱：info@quectel.com

或联系我司当地办事处，详情请登录：https://www.quectel.com.cn/contact。

如需技术支持或反馈我司技术文档中的问题，请随时登录网址：
https://www.quectel.com.cn/contact?tab=t 或发送邮件至：support@quectel.com。

前言

移远通信提供该文档内容以支持客户的产品设计。客户须按照文档中提供的规范、参数来设计产品。同时，您理解并同意，移远通信提供的参考设计仅作为示例。您同意在设计您目标产品时使用您独立的分析、评估和判断。在使用本文档所指导的任何硬软件或服务之前，请仔细阅读本声明。您在此承认并同意，尽管移远通信采取了商业范围内的合理努力来提供尽可能好的体验，但本文档和其所涉及服务是在“可用”基础上提供给您的。移远通信可在未事先通知的情况下，自行决定随时增加、修改或重述本文档。

使用和披露限制

许可协议

除非移远通信特别授权，否则我司所提供硬软件、材料和文档的接收方须对接收的内容保密，不得将其用于除本项目的实施与开展以外的任何其他目的。

版权声明

移远通信产品和本协议项下的第三方产品可能包含受移远通信或第三方材料、硬软件和文档版权保护的相关资料。除非事先得到书面同意，否则您不得获取、使用、向第三方披露我司所提供的文档和信息，或对此类受版权保护的资料进行复制、转载、抄袭、出版、展示、翻译、分发、合并、修改，或创造其衍生作品。移远通信或第三方对受版权保护的资料拥有专有权，不授予或转让任何专利、版权、商标或服务商标权的许可。为避免歧义，除了正常的非独家、免版税的产品使用许可，任何形式的购买都不可被视为授予许可。对于任何违反保密义务、未经授权使用或以其他非法形式恶意使用所述文档和信息的违法侵权行为，移远通信有权追究法律责任。

商标

除另行规定，本文档中的任何内容均不授予在广告、宣传或其他方面使用移远通信或第三方的任何商标、商号及名称，或其缩略语，或其仿冒品的权利。

第三方权利

您理解本文档可能涉及一个或多个属于第三方的硬软件和文档（“第三方材料”）。您对此类第三方材料的使用应受本文档的所有限制和义务约束。

移远通信针对第三方材料不做任何明示或暗示的保证或陈述，包括但不限于任何暗示或法定的适销性或特定用途的适用性、平静受益权、系统集成、信息准确性以及与许可技术或被许可人使用许可技术相关的不侵犯任何第三方知识产权的保证。本协议中的任何内容都不构成移远通信对任何移远通信产品或任何其他硬软件、设备、工具、信息或产品的开发、增强、修改、分销、营销、销售、提供销售或以其他方式维持生产的陈述或保证。此外，移远通信免除因交易过程、使用或贸易而产生的任何和所有保证。

隐私声明

为实现移远通信产品功能，特定设备数据将会上传至移远通信或第三方服务器（包括运营商、芯片供应商或您指定的服务器）。移远通信严格遵守相关法律法规，仅为实现产品功能之目的或在适用法律允许的情况下保留、使用、披露或以其他方式处理相关数据。当您与第三方进行数据交互前，请自行了解其隐私保护和数据安全政策。

免责声明

1) 移远通信不承担任何因未能遵守有关操作或设计规范而造成损害的责任。
2) 移远通信不承担因本文档中的任何因不准确、遗漏、或使用本文档中的信息而产生的任何责任。
3) 移远通信尽力确保开发中功能的完整性、准确性、及时性，但不排除上述功能错误或遗漏的可能。除非另有协议规定，否则移远通信对开发中功能的使用不做任何暗示或法定的保证。在适用法律允许的最大范围内，移远通信不对任何因使用开发中功能而遭受的损害承担责任，无论此类损害是否可以预见。
4) 移远通信对第三方网站及第三方资源的信息、内容、广告、商业报价、产品、服务和材料的可访问性、安全性、准确性、可用性、合法性和完整性不承担任何法律责任。

文档历史

修订记录

版本	日期	作者	变更描述
-	2023-12-14	Chapin Fang	文档创建
1.0	2024-09-02	Chapin Fang	受控版本
1.1	2025-12-08	Daly Deng	1. 新增适用模块 SH603ZA。 2. 新增 RKNN SDK 介绍（第 1 章）。 3. 代码块增加换行符提示。 4. 添加PDF 复制代码可能导致非预期换行的说明备注（第 1 章）。 5. 新增 RKNN 开发环境对应版本（第 2.1 章）。 6. 更新 RKNN 性能参考数据（第 2.4 章）。 7. 新增常见问题排查及相关测试设置方法（第 5 章）。

目录

表格索引

图片索引

1 引言

本文档主要介绍移远通信 QSM368ZP 设备及 SC362Z-AP、SG368Z 和 SH603ZA 模块的 RKNN（Rockchip Neural Network）使用方法。RKNN 是一款深度学习模型的各阶段流程一体化的开发和运行框架，它旨在对深度学习模型进行完整的开发流程支持，包括模型训练、模型优化、模型转换以及模型加载运行等。

RKNN SDK 是一套神经网络推理框架，能将 AI 计算任务分配到不同的硬件计算单元，如 CPU（Central Processing Unit）、NPU（Neural-network Processing Unit）等，充分利用 SoC 不同计算单元的所有算力，提高 AI 应用运行效率。此外，RKNN SDK 还支持自定义算子。

RKNN SDK 提供了一套私有、完整的深度学习推理框架，必须将 ONNX、Caffe 等模型转换为 SDK 支持的格式（*rknn*）才能在 NPU 环境下运行。RKNN SDK 执行了运行神经网络所需的大量繁重工作，为开发者节约了更多时间和资源，可以专注于构建其他创新的用户应用。

使用 RKNN，用户可以：

- 执行任意深度的神经网络；
- 可以加载到设备的 NPU 上进行计算；
- 在 x86 Ubuntu Linux 上训练网络模型；
- 将 Caffe、Caffe2、ONNX、Pytorch 和 TensorFlow 模型转换为 RKNN 深度学习容器（*rknn*）文件；
- 将 *rknn* 文件量化为 8 bit 定点，以便在 NPU 上运行；
- 通过 C++编程语言将网络集成到应用程序和其他代码中。

备注

1. 本文适用于运行 Linux 操作系统的 QSM368ZP 设备和运行 Android 或 Linux 操作系统的 SC362Z-AP、SG368Z 和 SH603ZA 模块。
2. 由于 PDF 格式特性，文档中部分代码块可能因页面宽度限制自动换行。直接从 PDF 复制代码到编辑器时，可能引入非预期的换行符（\n），导致代码无法正常运行。
   建议操作：
   若需从 PDF 复制，粘贴后请注意完成如下动作：
   - 手动删除复制粘贴后代码中的多余换行符。
   - 或通过代码编辑器的替换功能批量删除多余换行符（注意保留语义换行）。

2 RKNN SDK 架构

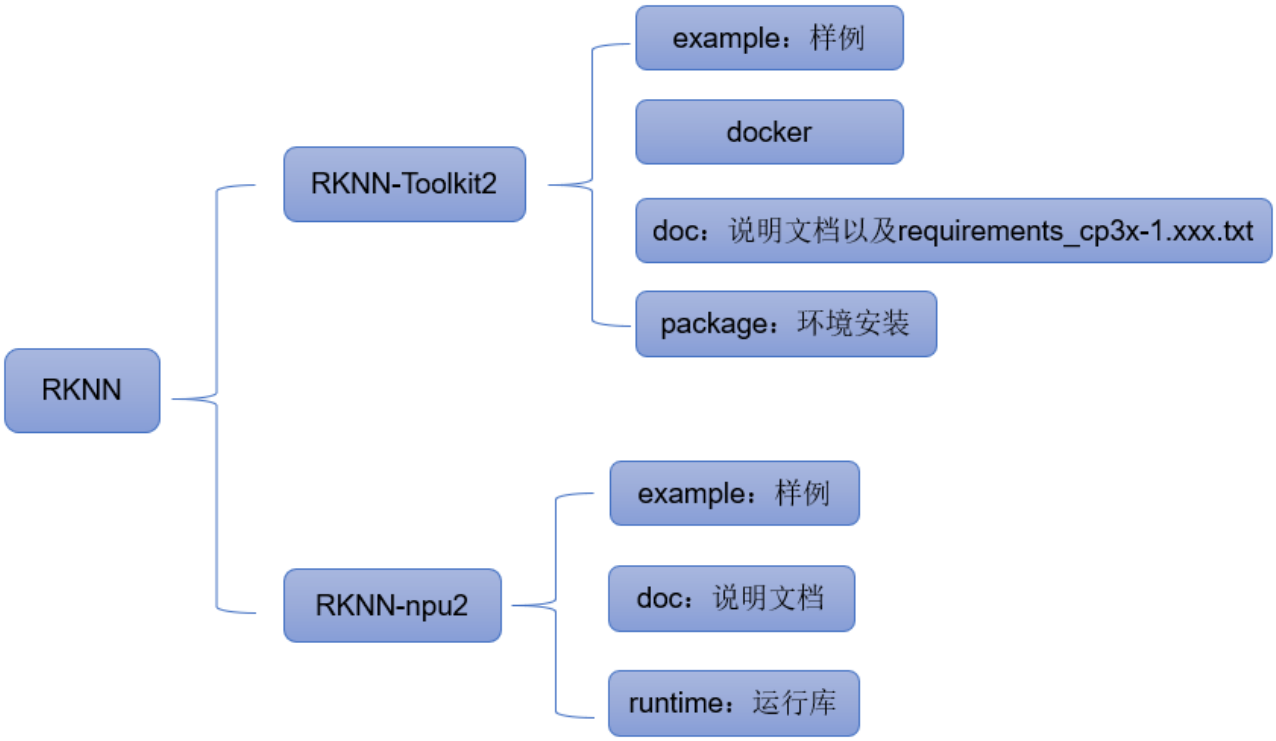


图 1：RKNN SDK 架构

2.1. RKNN 开发环境

1. RKNN SDK 可从 GitHub 网站下载：

   - rockchip-linux/rknn-toolkit2(github.com)
   - rockchip-linux/rknpu2(github.com)

2. 文档位置：

   - *RKNN-Toolkit2* 文档位于 *RKNN-Toolkit2* 包中的*<workspace>/doc* 目录
   - *RKNN-npu2* 文档位于 *RKNN-npu2* 包中的*<workspace>/doc* 目录

3. 环境要求：

目前，RKNN 1.52.0 及之前版本的 RKNN SDK 开发环境要求为：Ubuntu 18.04/Python 3.6、Ubuntu 20.04/Python 3.8 或 Ubuntu 22.04/Python 3.10。

SDK 开发前，需要事先准备 Caffe、TensorFlow、PyTorch 以及 ONNX 等环境。

表 1：开发环境对应版本

操作系统版本	Python 版本
Ubuntu 18.04 (x64)	3.6/3.7
Ubuntu 20.04 (x64)	3.8/3.9
Ubuntu 22.04 (x64)	3.10/3.11
Ubuntu 24.04 (x64)	3.12

具体的环境依赖及设置可查阅 SDK 中的相关文档：*<workspace>/doc/02_Rockchip_RKNPU_User_Guide_RKNN_SDK_V1.5.0_CN*。

2.2. RKNN SDK 文件目录结构

RKNN-Toolkit2 文件目录结构：

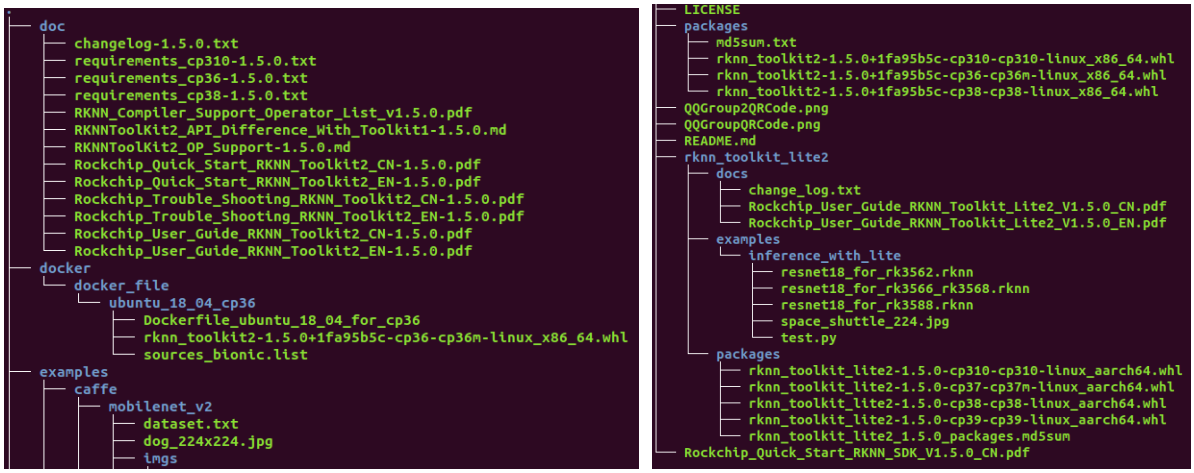


图 2：RKNN-Toolkit2 文件目录结构

RKNN-npu2 文件目录结构：

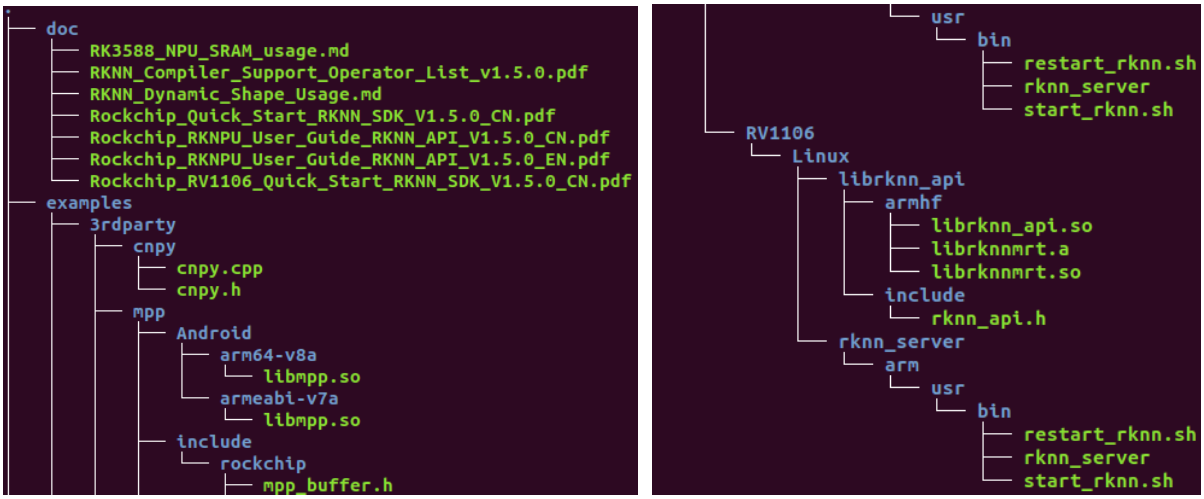


图 3：RKNN-npu2 文件目录结构

2.3. RKNN-Toolkit2 安装

下文以 Ubuntu 18.04/Python 3.6 为例，介绍 *RKNN-Toolkit2* 的安装与使用流程。

2.3.1. 环境准备

1. 一台装有 Ubuntu 18.04 操作系统的 x86_64 位的计算机；
2. RK356X EVB 开发板；
3. 将 EVB 板通过 USB 连接到 PC 上，使用 **adb devices** 命令查看连接结果，结果如下：

```
rk@rk:~$ adb devices
List of devices attached
515e9b401c060c0b        device
c3d9b8674f4b94f6        device
```

图 4：adb 连接结果

其中标红的是设备 ID。

2.3.2. 安装 RKNN-Toolkit2

1. 执行以下命令，安装 Python 3.6 和 pip3：

```
sudo apt-get install python3 python3-dev python3-pip
```

2. 执行以下命令，安装相关依赖：

```
sudo apt-get install libxslt1-dev zlib1g zlib1g-dev libglib2.0-0 libsm6 libgl1-mesa-glx \
libprotobuf-dev gcc
```

3. 获取并安装 *RKNN-Toolkit2* 安装包。

步骤 1： 进入 *RKNN-Toolkit2* 目录下打开终端执行以下命令，安装 Python 依赖：

```
pip install -r doc/requirements_cp36-1.x.x.txt
```

步骤 2： 执行以下命令，进入 *package* 目录：

```
cd package/
```

步骤 3： 执行以下命令，安装 *RKNN-Toolkit2*：

```
sudo pip install rknn_toolkit2-1.x.x+xxxxxxx-cp36-cp36m-linux_x86_64.whl
```

步骤 4： 执行以下命令，检查 *RKNN-Toolkit2* 是否安装成功：

```
python
from rknn.api import RKNN
```

如果导入 RKNN 模块没有失败，说明 *RKNN-Toolkit2* 安装成功。

2.4. RKNN 性能与精度

表 2：RKNN 性能数据

Model performance benchmark(FPS)										
demo	model_name	inputs_shape	dtype	RK3566 RK3568	RK3562	RK3588 @single_core	RK3576 @single_core	RV1109	RV1126	RK1808
mobilenet	mobilenetv2-12	[1, 3, 224, 224]	INT8	180.7	281.3	450.7	467	212.9	322.3	170.3
resnet	resnet50-v2-7	[1, 3, 224, 224]	INT8	37.9	54.9	110.1	99	24.4	36.2	37.1
yolov5	yolov5s_relu	[1, 3, 640, 640]	INT8	25.5	33.2	66.1	65	20.2	29.2	37.2
	yolov5n	[1, 3, 640, 640]	INT8	39.7	47.4	82.5	112.7	36.3	53.2	61.2
	yolov5s	[1, 3, 640, 640]	INT8	19.3	23.6	48.4	57.5	13.6	20	28.2
	yolov5m	[1, 3, 640, 640]	INT8	8.6	10.8	20.9	23.7	5.8	8.5	13.3
yolov6	yolov6n	[1, 3, 640, 640]	INT8	48.8	56.4	106.4	109.1	37.8	56.8	66.8
	yolov6s	[1, 3, 640, 640]	INT8	15.2	17.3	36.4	35	10.8	16.3	24.1
	yolov6m	[1, 3, 640, 640]	INT8	7.2	8.6	17.8	17.4	5.6	8.3	11.5
yolov7	yolov7-tiny	[1, 3, 640, 640]	INT8	27.9	36.5	72.7	74.8	15.4	22.4	37.2
	yolov7	[1, 3, 640, 640]	INT8	4.6	5.9	11.4	13	3.3	4.8	7.4
yolov8	yolov8n	[1, 3, 640, 640]	INT8	34	40.9	73.5	90.2	24	35.4	42.3
	yolov8s	[1, 3, 640, 640]	INT8	15.1	18.4	38	40.8	8.9	13.1	19.1
	yolov8m	[1, 3, 640, 640]	INT8	6.5	8.2	16.2	16.7	3.9	5.8	9.1
yolov8_obb	yolov8n-obb	[1, 3, 640, 640]	INT8	33.9	41.3	74	90.2	25.1	37.3	42.8
yolov10	yolov10n	[1, 3, 640, 640]	INT8	20.7	34.1	61.2	80.2	/	/	/
	yolov10s	[1, 3, 640, 640]	INT8	10.3	16.9	33.8	39.9	/	/	/
yolo11	yolo11n	[1, 3, 640, 640]	INT8	20.6	34	60	77.9	11.7	17	17.6
	yolo11s	[1, 3, 640, 640]	INT8	10.2	16.7	33	38.2	5	7.3	8.4
	yolo11m	[1, 3, 640, 640]	INT8	4.6	6.5	12.7	14.6	2.8	4	5.1
yolox	yolox_s	[1, 3, 640, 640]	INT8	15.2	18.3	37.1	41.5	10.6	15.7	23
	yolox_m	[1, 3, 640, 640]	INT8	6.6	8.2	16	17.6	4.6	6.8	10.7
ppyoloe	ppyoloe_s	[1, 3, 640, 640]	INT8	17.1	20	32.5	41.3	11.2	16.4	21.1
	ppyoloe_m	[1, 3, 640, 640]	INT8	7.8	9.2	15.8	17.8	5.2	7.7	9.4
yolo_world	yolo_world_v2s	[1, 3, 640, 640]	INT8	7.4	9.6	22.1	22.3	/	/	/
	clip_text	[1, 20]	FP16	29.8	67.4	95.8	63.5	/	/	/
yolov8_pose	yolov8n-pose	[1, 3, 640, 640]	INT8	22.6	31	55.9	66.8	/	/	/
deeplabv3	deeplab-v3-plus-mobilenet-v2	[1, 513, 513, 1]	INT8	10.9	21.4	34	39.4	10.1	13	4.4
yolov5_seg	yolov5n-seg	[1, 3, 640, 640]	INT8	32.2	38.5	69.3	88.3	28.6	42.2	49.6
	yolov5s-seg	[1, 3, 640, 640]	INT8	15	18.1	36.8	41.6	9.6	14	22.5
	yolov5m-seg	[1, 3, 640, 640]	INT8	6.8	8.4	16.4	18	4.7	6.8	10.8
yolov8_seg	yolov8n-seg	[1, 3, 640, 640]	INT8	27.8	33	60.8	71.1	18.6	27.6	32.9
	yolov8s-seg	[1, 3, 640, 640]	INT8	11.7	14.1	28.9	30.8	6.6	9.8	14.6
	yolov8m-seg	[1, 3, 640, 640]	INT8	5.2	6.4	12.6	12.7	3.1	4.6	6.9
ppseg	ppseg_lite_1024x512	[1, 3, 512, 512]	INT8	5.9	13.9	35.7	33.6	18.4	27.1	20.9
mobilesam	mobilesam_encoder_tiny	[1, 3, 448, 448]	FP16	1	6.6	10	11.9	/	/	/
	mobilesam_decoder	[1, 1, 112, 112]	FP16	24.3	69.6	116.4	108.6	/	/	/
RetinaFace	RetinaFace_mobile320	[1, 3, 320, 320]	INT8	156.4	300.8	227.2	470.5	144.8	212.5	198.5
	RetinaFace_resnet50_320	[1, 3, 320, 320]	INT8	18.7	26.9	49.2	56.6	14.6	20.8	24.6
LPRNet	lprnet	[1, 3, 24, 94]	FP16	143.2	420.6	586.4	647.8	30.6(INT8)	47.6(INT8)	30.1(INT8)
PPOCR-Det	ppocrv4_det	[1, 3, 480, 480]	INT8	22.1	28	50.7	64.3	11	16.1	14.2
PPOCR-Rec	ppocrv4_rec	[1, 3, 48, 320]	FP16	19.5	54.3	73.9	96.8	1	1.6	6.7
lite_transformer	lite-transformer-encoder-16	embedding-256, token-16	FP16	337.5	725.8	867.6	784.1	22.7	35.4	98.3
	lite-transformer-decoder-16	embedding-256, token-16	FP16	142.5	252	343.8	272.3	48	65.8	109.9
clip	clip_images	[1, 3, 224, 224]	FP16	2.3	3.4	6.5	6.7	/	/	/
	clip_text	[1, 20]	FP16	29.7	66.6	96	63.7	/	/	/
wav2vec2	wav2vec2_base_960h_20s	20s audio	FP16	RTF 0.817	RTF 0.323	RTF 0.133	RTF 0.073	/	/	/
whisper	whisper_base_20s	20s audio	FP16	RTF 1.178	RTF 0.42	RTF 0.215	RTF 0.218	/	/	/
zipformer	zipformer-bilingual-zh-en-t	streaming audio	FP16	RTF 0.196	RTF 0.116	RTF 0.065	RTF 0.082	/	/	/
yamnet	yamnet_3s	3s audio	FP16	RTF 0.013	RTF 0.008	RTF 0.004	RTF 0.005	/	/	/
mms_tts	mms_tts_eng_200	token-200	FP16	RTF 0.311	RTF 0.138	RTF 0.069	RTF 0.069			

备注

1. 该数据为 *rknn_model_zoo* 中所提供数据。
2. 该性能数据基于各平台的最大 NPU 频率进行测试。
3. 该性能数据指模型推理的耗时，不包含前后处理的耗时。
4. “/”表示当前版本暂不支持。

2.5. RKNN 支持的网络层

operator	支持情况	输入数据类型	输入	设置项/ 输入参数含义	约束规格	广播支持（维度补齐）	量化支持方式
Add/Bias	支持	int8 float16	input_tensor [batch, channel, height, width]:tensor	batch/ 输入的batch channel/ 输入的channel height/ 输入的height width/ 输入的width	无限制	支持ONNX规范的四维tensor的所有广播操作，以ONNX默认排列NCHW做说明，支持以下广播方式：1.OP(A(N,C,H,W),B(N,C,H,W))，即两个维度相同的 tensor进行操作 2. OP(A(N,C,H,W),B(C,1,1))，即C维度做broadcasting 3. OP(A(N,C,H,W),B(scalar))，即以单个标量做broadcasting 说明：A或B都可以作为广播方。	per-layer/ per-channel
Sub	支持	int8 float16	input_tensor [batch, channel, height, width]:tensor	batch/ 输入的batch channel/ 输入的channel height/ 输入的height width/ 输入的width	无限制	支持两个tensor的广播操作，以ONNX默认排列NCHW做说明，支持以下广播方式： 1.OP(A(N,C,H,W),B(N,C,H,W))，即两个维度相同的 tensor进行操作 2. OP(A(N,C,H,W),B(C,1,1))，即C维度做broadcasting 3. OP(A(N,C,H,W),B(scalar))，即以单个标量做broadcasting 说明：A或B都可以作为广播方。	per-layer/ per-channel
Mul/Scale	支持	int8 float16	input_tensor [batch, channel, height, width]:tensor	batch/ 输入的batch channel/ 输入的channel height/ 输入的height width/ 输入的width	无限制	支持ONNX规范的四维tensor的所有广播操作，以ONNX默认排列NCHW做说明，支持以下广播方式：1.OP(A(N,C,H,W),B(N,C,H,W))，即两个维度相同的 tensor进行操作 2. OP(A(N,C,H,W),B(C,1,1))，即C维度做broadcasting 3. OP(A(N,C,H,W),B(scalar))，即以单个标量做broadcasting 4. OP(A(N,C,H,W),B(H,W))，即HW维度做broadcasting，目前仅支持FP16类型 说明：A或B都可以作为广播方。	per-layer/ per-channel

operator	支持情况	输入数据类型	输入	设置项/ 输入参数含义	约束规格	广播支持（维度补齐）	量化支持方式
Global AveragePool	支持	int8 float16	input_tensor [batch, channel, height, width]:tensor	batch/ 输入的batch	1		per-layer
				channel/ 输入的channel	[1,8192]		
				height/ 输入的height	[1,343]（toolkit2支持范围）		
				width/ 输入的width			
GlobalMaxPool	支持	int8 float16	input_tensor [batch, channel, height, width]:tensor	batch/ 输入的batch	1		per-layer
				channel/ 输入的channel	[1,8192]		
				height/ 输入的height	[1,343]（toolkit2支持范围）		
				width/ 输入的width			

图 5：RKNN 支持的网络层

备注

不同版本 SDK 所支持的网络层不同，详细内容请查阅 SDK 中的相关文档：*doc/RKNN_Compiler_Support_Operator_List_v1.5.0*。

3 RKNN 工作流程

3.1. 模型运行流程

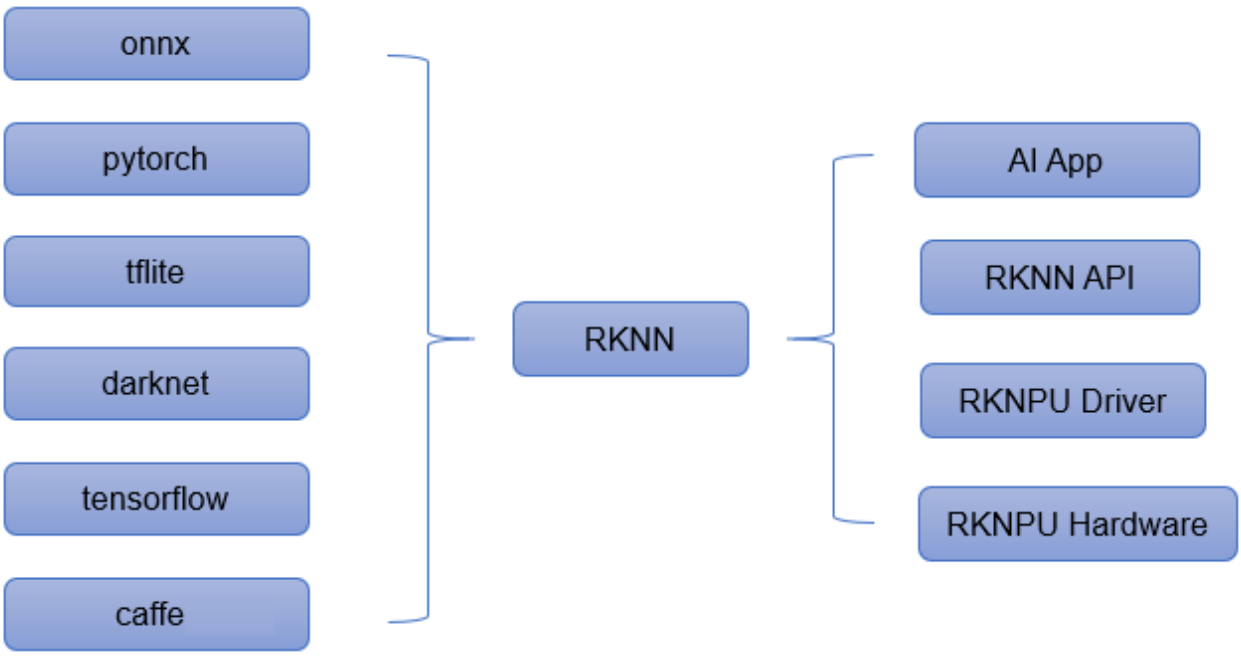


图 6：模型运行流程

3.2. RKNN 基本工作流程

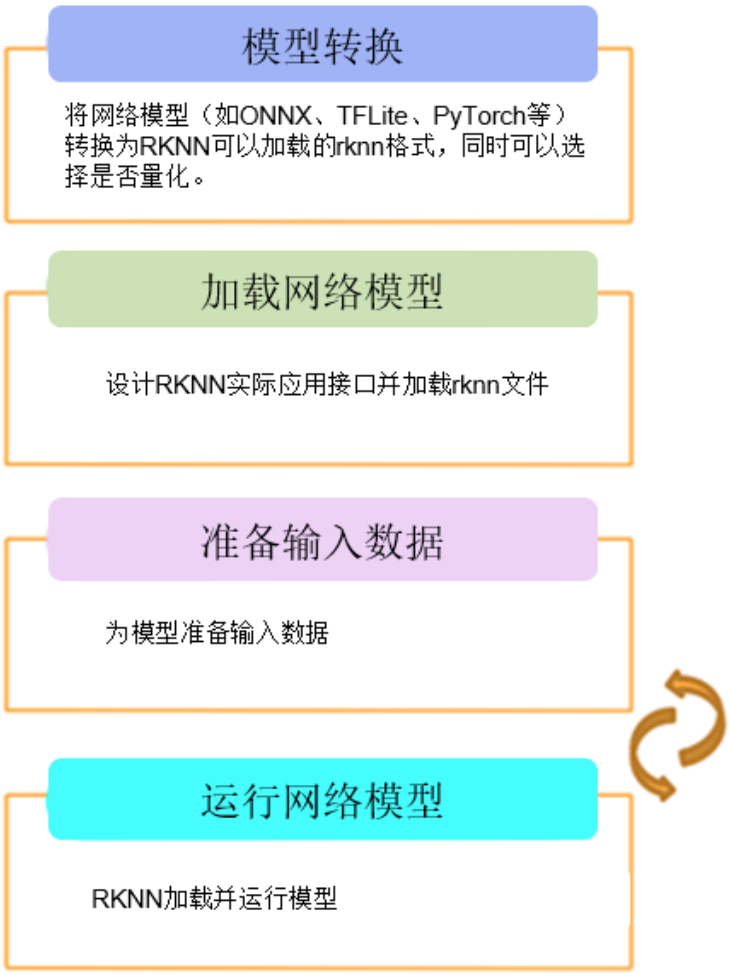


图 7：RNKK 基本工作流程

3.2.1. 模型转换

使用 *rknn-toolkit2/examples/onnx/yolov5/test.py* 将网络模型转为 RKNN：

在 *test.py* 中：

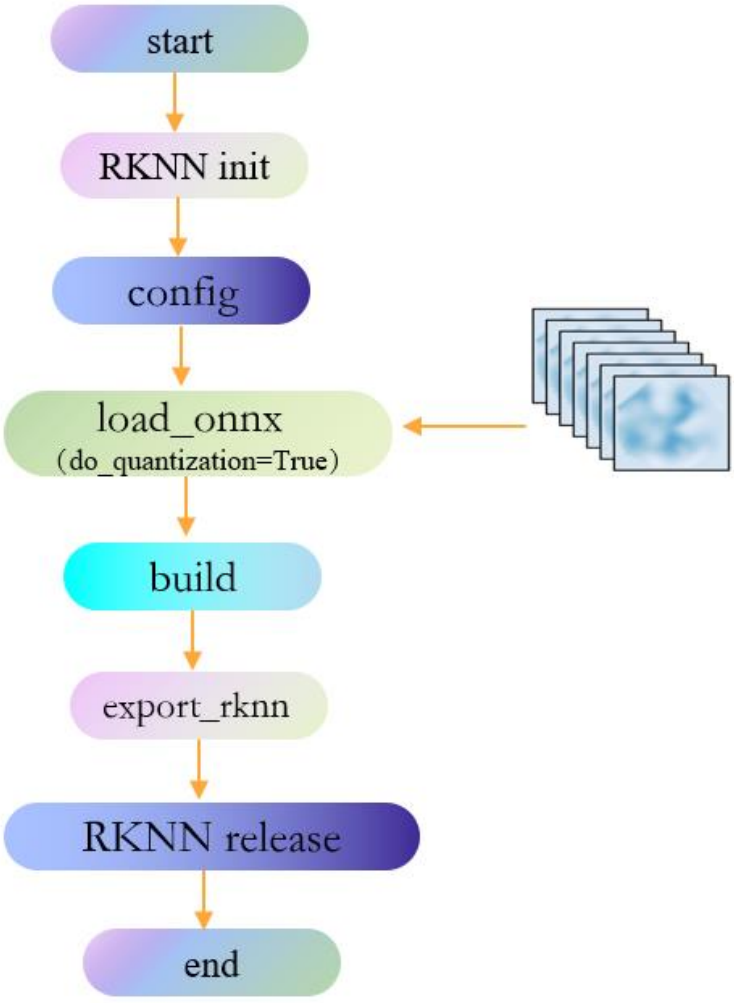
- 使用 *rknn.load_onnx* 可将基于 ONNX 的网络模型转为 RKNN；
- 使用 *rknn.load_caffe* 可将基于 Caffe 的网络模型转为 RKNN；
- 使用 *rknn.load_tflite* 可将基于 TFLite 的网络模型转为 RKNN；
- 使用 *rknn.load_pytorch* 可将基于 PyTorch 的网络模型转为 RKNN；
- 使用 *rknn.load_darknet* 可将基于 Darknet 的网络模型转为 RKNN。

有关转换工具和语法的其他详细信息，请查阅 SDK 中的相关文档：*doc/Rockchip_User_Guide_RKNN_Toolkit2_CN-1.5.0*。

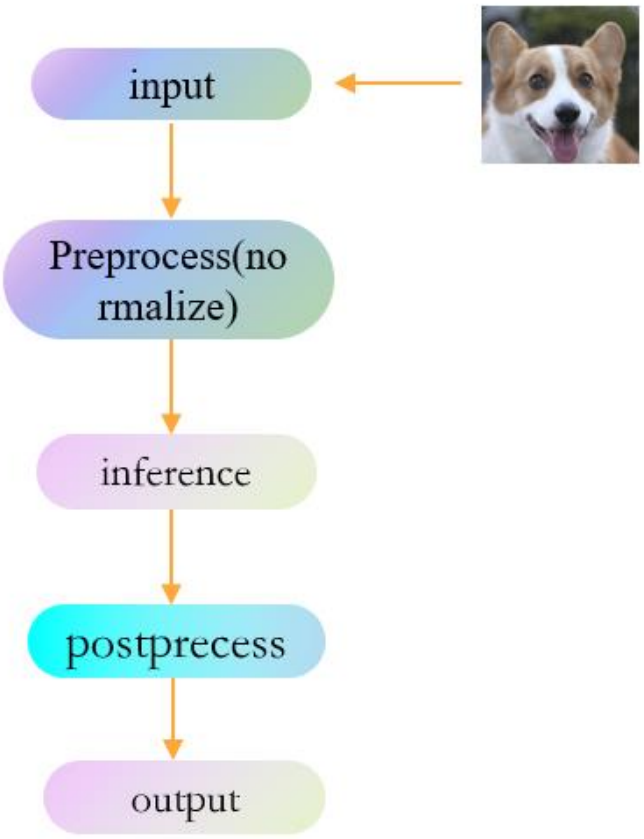
3.2.2. 模型量化

使用 *rknn-toolkit2/examples/onnx/yolov5/test.py* 将网络模型转为 RKNN 时，可以在 *rknn.build(do_quantization=QUANTIZE_ON, dataset=DATASET)*中选择 *quantization* 为 *True* 或 *False*。

步骤 1： 量化一个模型（ONNX 模型）。



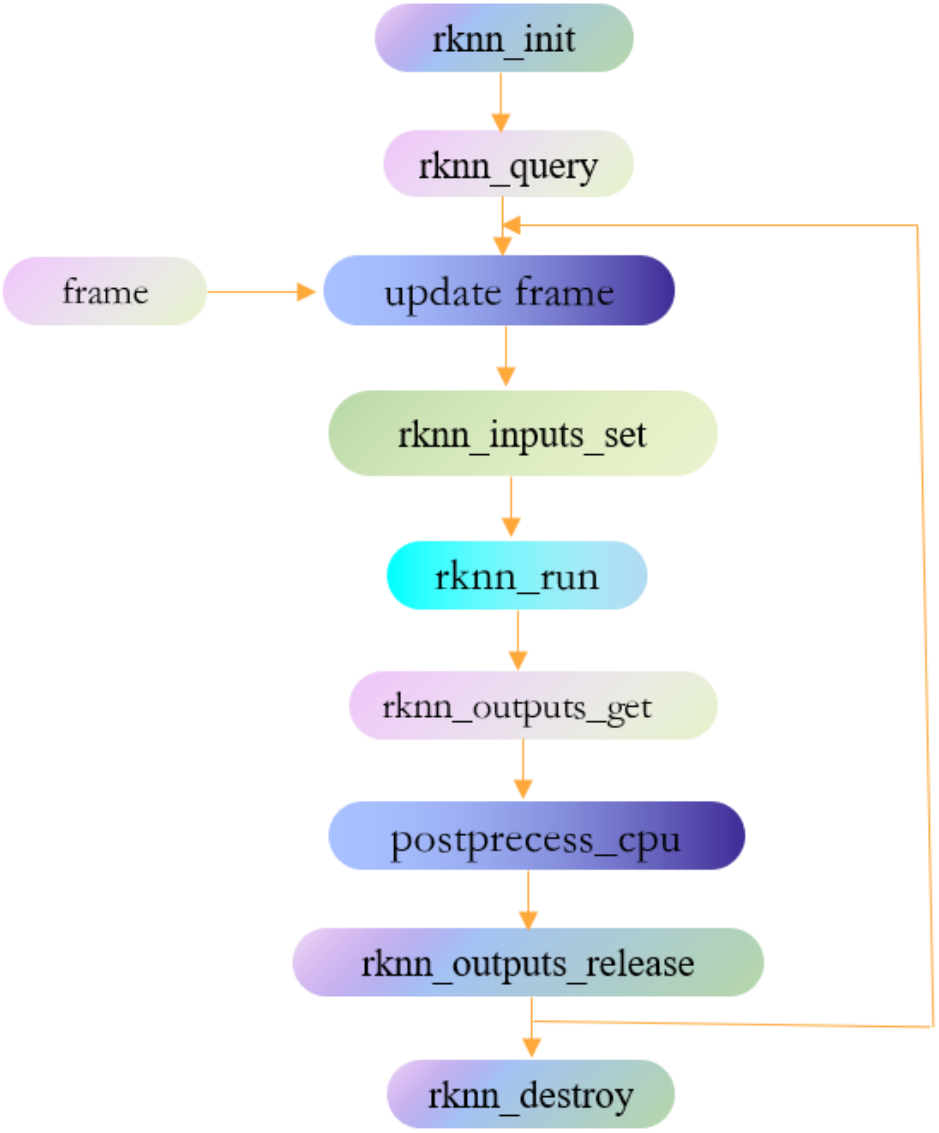
步骤 2： 在网络创建后，通过网络运行一个图像。



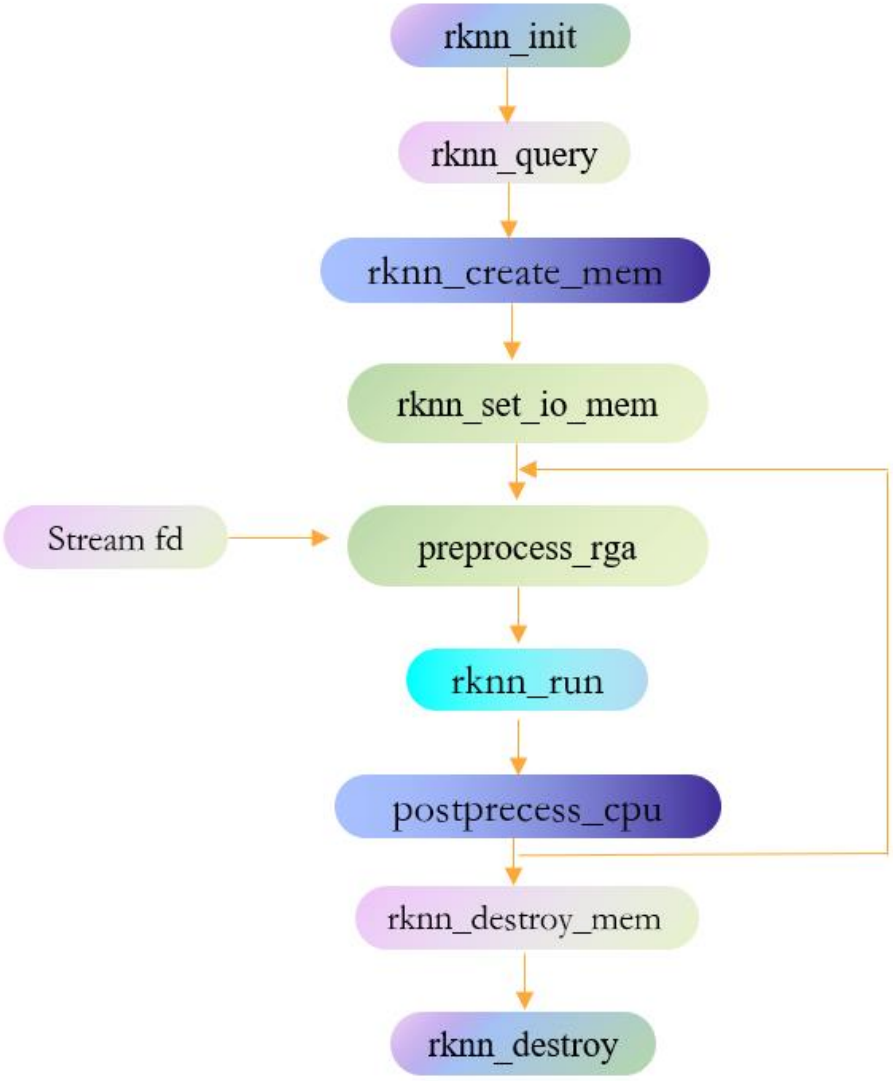
3.2.3. 网络模型加载

RKNN SDK 有两组 API 可以使用，分别是通用 API 以及零拷贝 API。两组 API 的主要区别在于，通用 API 每次更新数据，需要将外部模块分配的数据拷贝到 NPU 运行时的输入内存，而零拷贝 API 会直接使用预先分配的内存。通用 API 和零拷贝 API 不能混合使用。

1. 通用 API，首先初始化 *rknn_input* 结构体，帧数据包含在该结构体中，使用 *rknn_input_set* 函数设置模型输入，使用 *rknn_outputs_get* 函数获取推理的输出，进行后处理。在每次推理前，更新帧数据。通用 API 调用流程如下图所示：



2. 对于零拷贝 API，在分配内存后，使用内存信息初始化 *rknn_tensor_mem* 结构体，在推理前创建并设置该结构体，并在推理后读取该结构体中的内存信息。



3.2.4. 输入数据准备

1. RKNN 要求输入图像是不同于 Caffe 或 Tensorflow 网络的特定格式；

2. 在推理过程中使用的通道顺序必须与在训练过程中使用的通道顺序相同。例如，在 Caffe 中训练的图像集模型需要一个 BGR 的通道顺序；

3. 对于 RKNN，对于使用减去平均值的图像进行训练的网络模型，必须在 *rknn.config* 中添加 *mean_values* 和 *std_values*；

4. 有关输入数据处理的其他详细信息，请查阅 SDK 中相关文档：*doc/Rockchip_User_Guide_RKNN_Toolkit2_CN-1.5.0*。

4 Android/Linux 系统 RKNN 使用实例

使用 RKNN SDK 中的实例模型转换运行，需先在 PC 环境进行模型转换，转换成 RKNN 模型，然后 push 到设备环境中推理，注意需要根据系统选用不同的执行文件和库。

RKNN 官方提供一系列使用实例，在 *rknpu2/examples/rknn_yolov5_demo* 文件夹内。以下以 *yolov5* 为例，介绍基本操作及使用。

4.1. 准备工作

- Ubuntu 18.04
- Python 3.6
- onnx 1.10.0
- RKNN 1.5.0
- Android：ANDROID_NDK = 18rb
  Linux：GCC = gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu
- RKNN3568 EVB 板
- USB 线

4.2. 环境及依赖设置

1. 安装 adb：

   执行如下命令，安装和破解 adb：

   ```
   sudo apt-get install android-tools-adb
   ```

   每次主机连接 EVB 板后，需要执行以下命令，方可进入模块环境：

   ```
   adb root
   adb remount
   adb shell
   ```

2. 设置 SDK 环境：

   用户可以从 GitHub 下载的 *RKNN-Toolkit2* 以及 *RKNN-npu2* 中获得相关文件夹。下面以 *rknn-1.5.0* 为例说明如何设置 SDK 环境。

   - 执行 **pip install -r requirements_cp36-1.5.0.txt** 命令安装所需要的依赖。
   - 从 https://developer.android.google.cn/ndk/downloads?hl=zh-cn 下载 *android-ndk-r18b*，用于 Android 平台。
   - 从 Linaro Releases 下载 *gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu*，用于 Linux 平台。

4.3. 在 Ubuntu 18.04 主机上转换模型

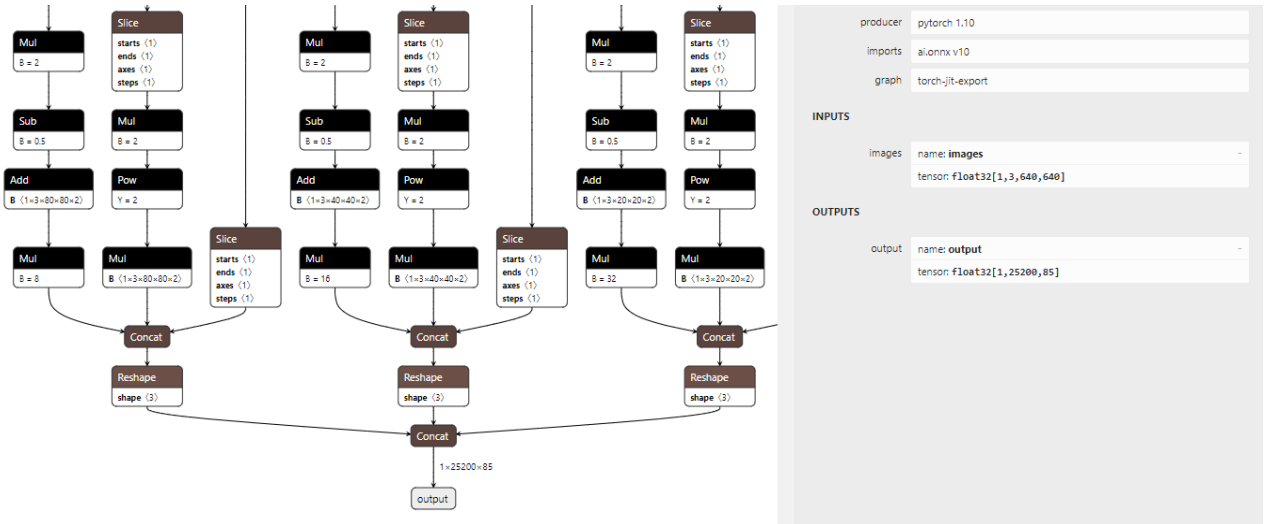
1. 获取 *yolov5* Model。

   通过如下路径下载 *yolov5*：

   ultralytics/yolov5: YOLOv5 🚀 in PyTorch > ONNX > CoreML > TFLite (github.com)

   运行 *export.py* 脚本获得 *yolov5s.onnx*，此时获得是只有一个头输出的 ONNX 模型，如下图所示。

   执行 **python export.py --weights weights/yolov5s.pt --imgsz 640 --opset 11 --include onnx** 将模型从 pt 转换为 ONNX。



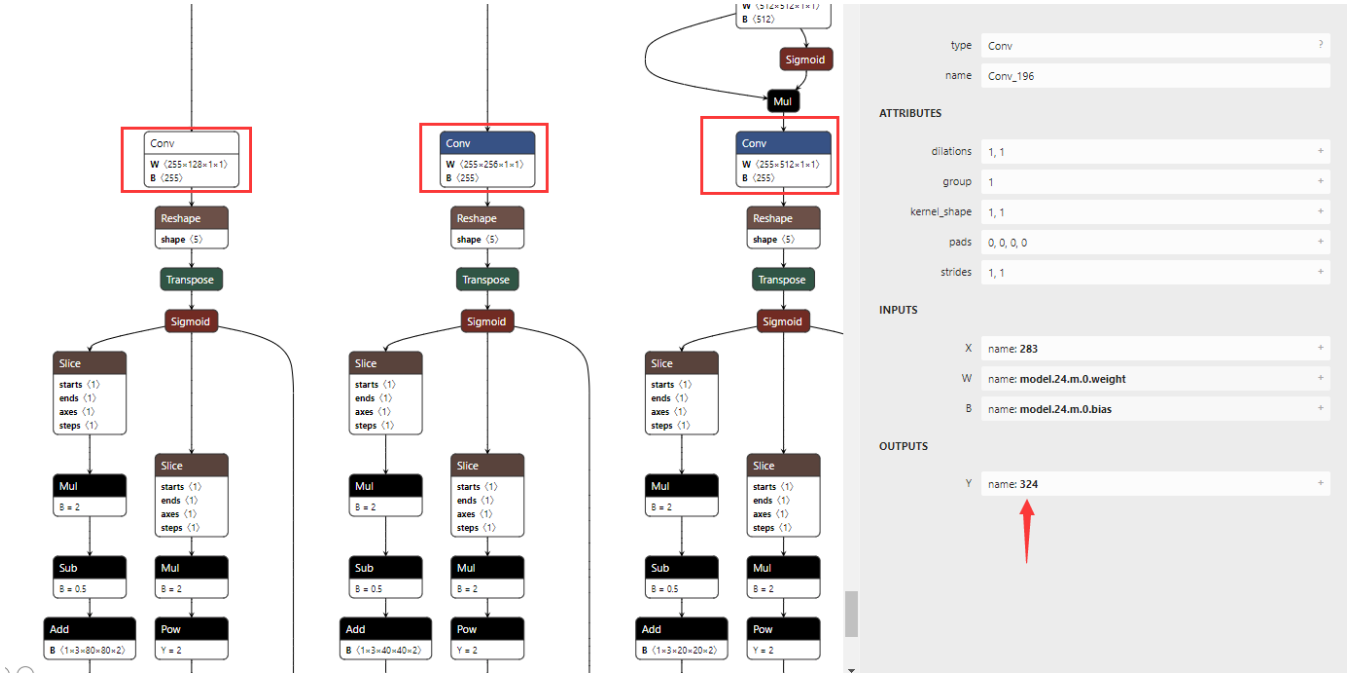
2. 转为 RKNN 模型。

   进入 *rknn-toolkit2/examples/onnx/yolov5*，打开 *test.py* 设置 ONNX 路径、RKNN 保存路径以及 *dataset.txt* 用于模型量化。将 *rknn.load_onnx* 中的 *outputs* 参数设置为三个卷积层的输出，因为模块并不支持五维运算。如下所示：

```
# pre-process config
print('--> Config model')
rknn.config(mean_values=[[0, 0, 0]], std_values=[[255, 255, 255]])
print('done')

# Load ONNX model
print('--> Loading model')
ret = rknn.load_onnx(model=ONNX_MODEL,outputs=['324', '377', '430'])
if ret != 0:
    print('Load model failed!')
    exit(ret)
print('done')
```

*outputs=['324', '377', '430']*在 ONNX 中如下图所示：



4.4. 在 Android 设备上运行 yolov5

进入 *rknpu2/examples/rknn_yolov5_demo*，将***第 4.3 章***生成的 RKNN 模型放进该目录下，打开该目录下的 *build-android_RK3566_RK3568.sh*，设置 *ANDROID_NDK_PATH*，如下图所示：

```
set -e

if [ -z ${ANDROID_NDK_PATH} ]
then
  ANDROID_NDK_PATH=/home/zdl/AndroidSDK/android-ndk-r18b
fi

BUILD_TYPE=Release

TARGET_SOC="rk356x"

ROOT_PWD=$( cd "$( dirname $0 )" && cd -P "$( dirname "$SOURCE" )" && pwd )
```

设置完成之后，打开终端执行**./ build-android_RK3566_RK3568.sh**。编译完成之后，会出现 *install* 文件夹。将 *install* 文件下 *rknn_yolov5_demo_Android* push 到 EVB 板中，同时将转换完成的 *yolov5s.rknn* 模型和待测图像 push 到 EVB 板中，具体步骤如下所示：

步骤 1：

```
cd examples/rknn_yolov5_demo
```

步骤 2：

```
./ build-android_RK3566_RK3568.sh
```

步骤 3：

```
adb push install/rknn_yolov5_demo_Android ./data
adb push yolov5s.rknn ./data/rknn_yolov5_demo
adb push bus.jpg ./data/rknn_yolov5_demo
```

步骤 4：

```
adb shell
cd    data/rknn_yolov5_demo
export LD_LIBRARY_PATH=./lib
./rknn_yolov5 yolov5.rknn bus.jpg
```

最终效果展示：

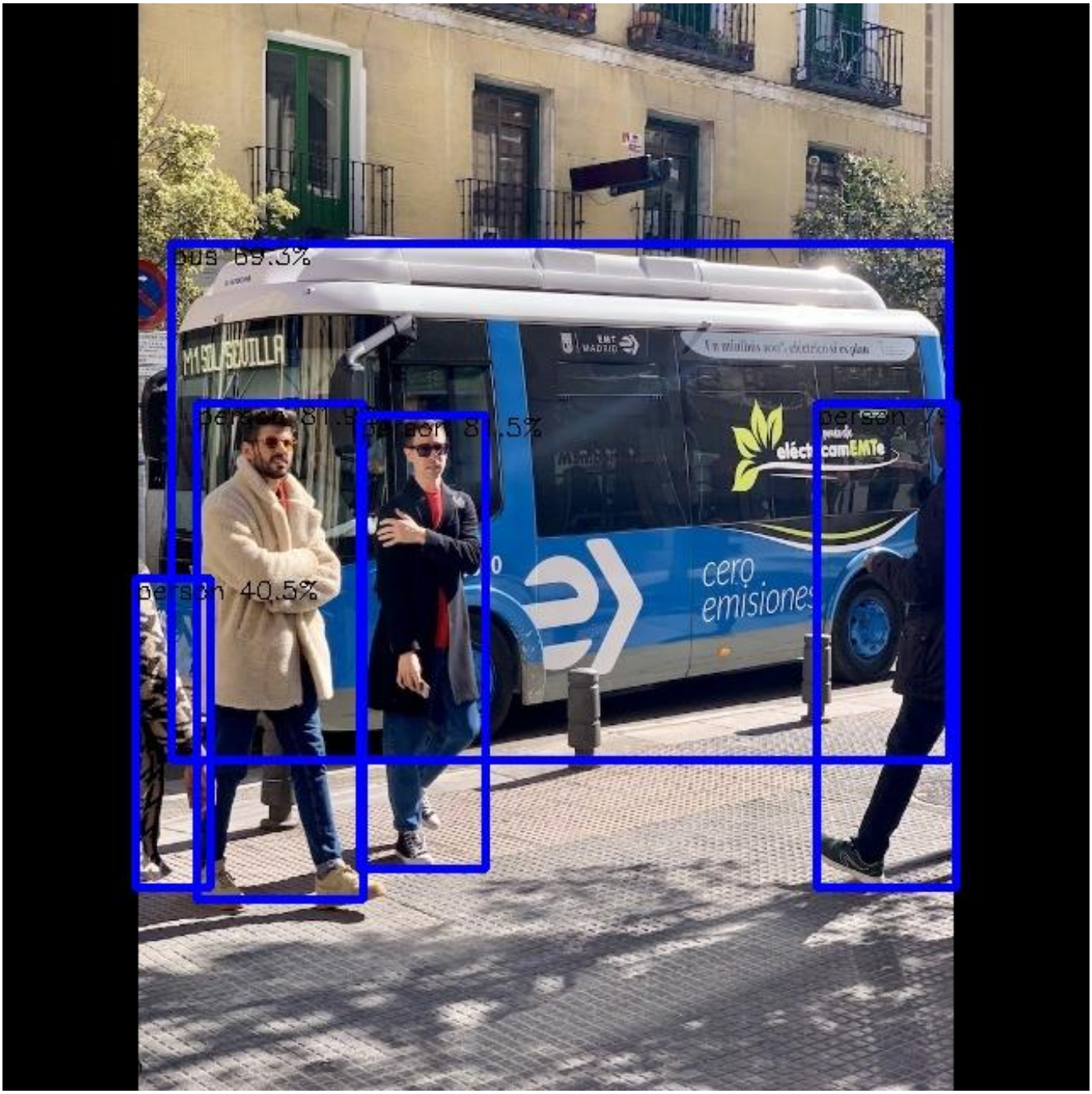


图 8：Android 设备上运行 yolov5（最终效果）

4.5. 在 Linux 设备上运行 yolov5

进入 *rknpu2/examples/rknn_yolov5_demo*，将***第 4.3 章***生成的 RKNN 模型放进该目录下，打开该目录下的 *build-linux_RK356X.sh*，设置 *GCC_COMPILER*，如下图所示：

```
set -e

TARGET_SOC="rk356x"
#GCC_COMPILER=aarch64-linux-gnu
GCC_COMPILER=/home/zdl/rk3568/gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu
```

设置完成之后，打开终端执行**./ build-linux_RK356X.sh**。编译完成之后，会出现 *install* 文件夹。将 *install* 文件下 *rknn_yolov5_demo_linux* push 到 EVB 板中，同时将转换完成的 *yolov5s.rknn* 模型和待测图像 push 到 EVB 板中，具体步骤如下所示：

步骤 1：

```
cd examples\rknn_yolov5_demo
```

步骤 2：

```
./build-linux_RK356X.sh
```

步骤 3：

```
adb push install/rknn_yolov5_demo_Linux ./data
adb push yolov5s.rknn ./data/rknn_yolov5_demo
adb push bus.jpg ./data/rknn_yolov5_demo
```

步骤 4：

```
adb  shell
cd  data/rknn_yolov5_demo
export  LD_LIBRARY_PATH=./lib
./rknn_yolov5 yolov5.rknn bus.jpg
```

最终效果展示：

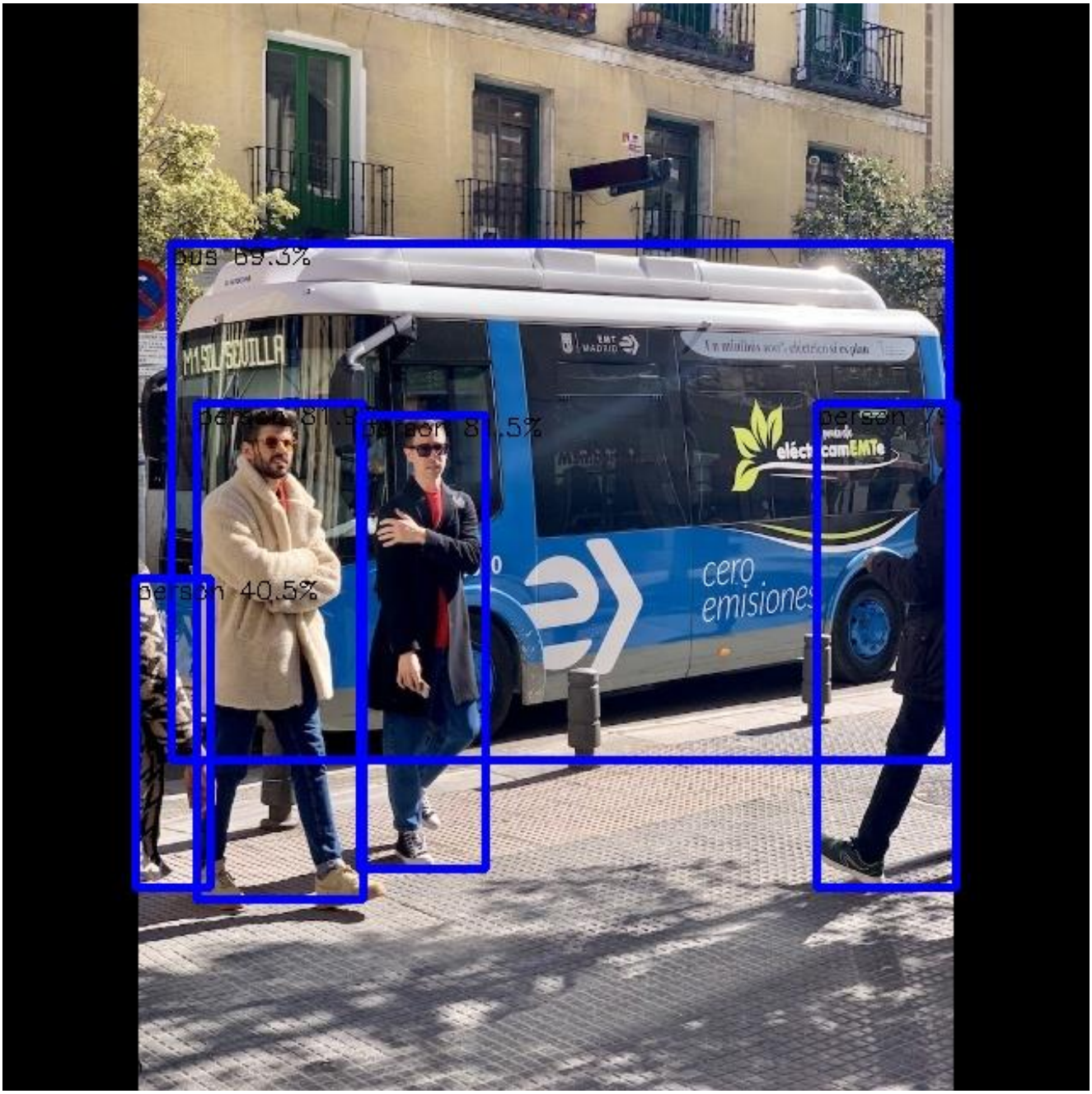


图 9：Linux 设备上运行 yolov5（最终效果）

5 常见问题

5.1. 执行 adb devices 命令无法查看设备

可以尝试以下方式来解决此问题：

- 检查连线是否正确
- 重新插拔数据线
- 换计算机另一个 USB 端口来连接数据线
- 更换数据线

出现以下错误时，表示系统中未安装 adb。需要执行命令 **apt install adb** 安装 adb。

```
command 'adb' not found, but can be installed with:
apt install adb
```

5.2. 编译阶段报错或运行时提示找不到某些依赖库

这是由于当前使用的交叉编译器和 *rknn_model_zoo* 默认使用的交叉编译器不同所导致，请参考 *rknn_model_zoo/docs/Compilation_Environment_Setup_Guide_CN.md* 文档说明使用对应编译器重新编译。

备注

重新编译前需删除之前编译的 *build* 目录。

5.3. 查看在 EVB 开发板上运行推理时 CPU/GPU/NPU 的占用率

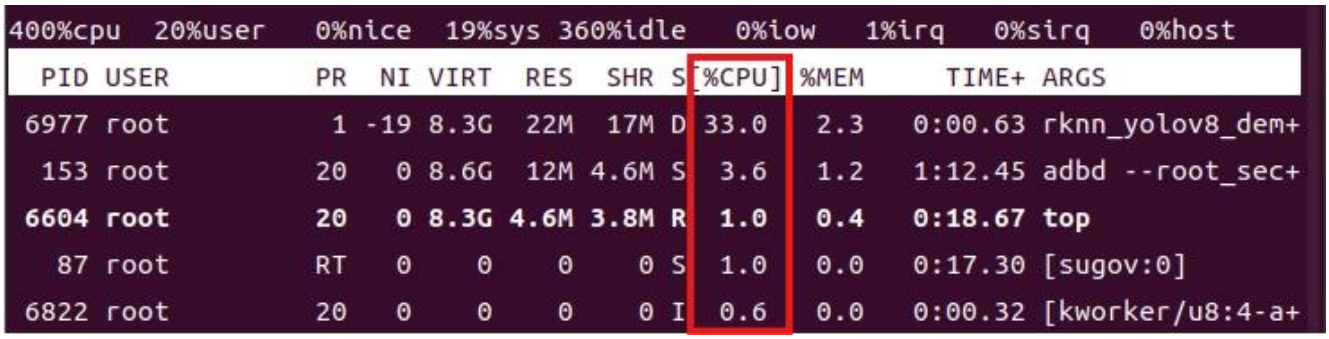
本章节以 SH603ZA 模块为例进行说明。

5.3.1. 查看 CPU 占用率

打开两个窗口，一个窗口用于执行程序，另一个窗口用于查看占用率（GPU/NPU 同理），执行如下命令查看 CPU 占用率：

```
adb shell
top
```

输出结果如下：



5.3.2. 查看 GPU 占用率

执行如下命令查看 GPU 设备节点的正确路径：

```
adb shell
#搜索所有 GPU 相关的 devfreq 路径
find /sys -name "*gpu*" -path "*/devfreq/*" 2>/dev/null
```

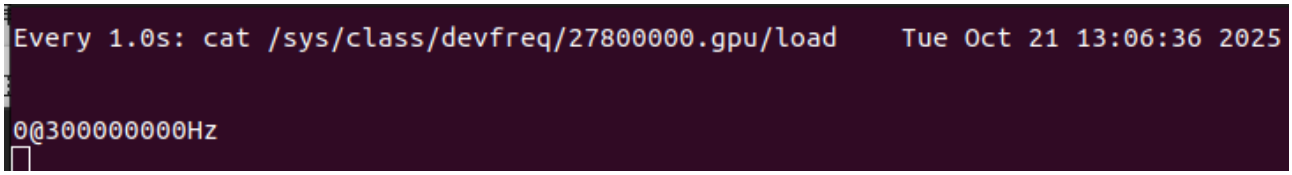
输出结果如下：

如上图所示，SH603ZA 模块的 GPU 节点为 27800000。

根据查询到的对应 GPU 节点，执行如下命令查看 GPU 占用（每一秒刷新一次）：

```
watch -n 1 cat /sys/class/devfreq/27800000.gpu/cur_freq
```

输出结果如下：

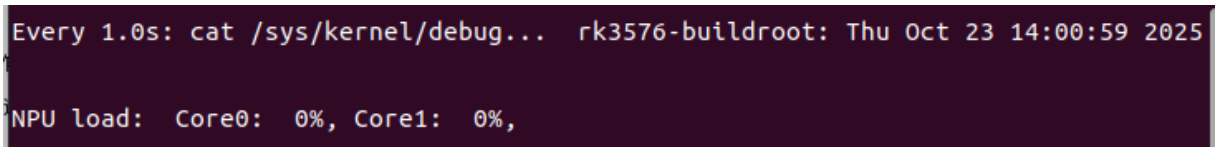


5.3.3. 查看 NPU 占用率

执行如下命令查看 NPU 占用率：

```
adb shell
#实时监控 NPU 的占用率，可以使用 watch 命令（每一秒刷新一次）
watch -n 1 cat /sys/kernel/debug/rknpu/load
```

输出结果如下，其中 NPU load 即为占用率：



5.4. 查询和设置 CPU/GPU/NPU 频率

本章节以 SH603ZA 模块为例进行说明。

执行如下命令查询当前 CPU 频率：

```
cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_cur_freq
```

执行如下命令查询当前 GPU 频率：

```
cat /sys/class/devfreq/27800000.gpu/cur_freq
```

执行如下命令查询当前 NPU 频率：

```
cat /sys/class/devfreq/27700000.npu/cur_freq
```

执行如下命令查询当前 DDR 频率：

```
cat /sys/class/devfreq/dmc/cur_freq
```

5.4.1. CPU 定频

步骤 1： 执行如下命令查看 CPU 可用频率（以 policy0 为例）：

```
cat /sys/devices/system/cpu/cpufreq/policy0/scaling_available_frequencies
```

输出结果如下：

```
408000 600000 816000 1008000 1200000 1416000 1608000 1800000 2016000
```

步骤 2： 执行如下命令手动切换 userspace 模式：

```
echo userspace > /sys/devices/system/cpu/cpufreq/policy0/scaling_governor
```

步骤 3： 执行如下命令设置频率为 600 MHz：

```
echo 600000 > /sys/devices/system/cpu/cpufreq/policy0/scaling_setspeed
```

步骤 4： 设置完成后，可再次查看 CPU 频率确认是否设置成功。

5.4.2. GPU 定频

步骤 1： 执行如下命令查看 GPU 可用频率。

```
cat /sys/class/devfreq/27800000.gpu/available_frequencies
```

输出结果如下：

```
900000000 800000000 700000000 600000000 500000000 400000000 300000000
```

步骤 2： 执行如下命令手动切换 userspace 模式。

```
echo userspace > /sys/class/devfreq/27800000.gpu/governor
```

步骤 3： 执行如下命令设置频率 400 MHz。

```
echo 400000000 > /sys/class/devfreq/27800000.gpu/userspace/set_freq
```

步骤 4： 设置完成后，可再次查看 GPU 频率确认是否设置成功。

5.4.3. NPU 定频

步骤 1： 执行如下命令查看 NPU 可用频率。

```
cat /sys/class/devfreq/27700000.npu/available_frequencies
```

输出结果如下：

```
300000000 400000000 500000000 600000000 700000000 800000000 900000000 950000000
```

步骤 2： 执行如下命令手动切换 userspace 模式。

```
echo userspace > /sys/class/devfreq/27700000.npu/governor
```

步骤 3： 执行如下命令设置频率为 900 MHz。

```
echo 900000000 > /sys/class/devfreq/27700000.npu/userspace/set_freq
```

步骤 4： 设置完成后可再次查看 NPU 频率确认是否设置成功。

5.5. 设置单/双核（NPU）在 EVB 开发板上运行（仅适用 SH603ZA 模块）

RKNN 提供了 *rknn_set_core_mask()* API，该函数指定工作的 NPU 核心，若在单核 NPU 架构的平台上设置会返回错误。

*rknn_set_core_mask()*参数说明如下：

类型	参数	描述
rknn_context	*context*	rknn_context 对象
rknn_core_mask	*core_mask*	NPU 核心的枚举类型，目前有如下方式配置： *RKNN_NPU_CORE_AUTO* 自动调度模型，自动运行在当前空闲的 NPU 核上 *RKNN_NPU_CORE_0* 运行在 NPU 0 核上 *RKNN_NPU_CORE_1* 运行在 NPU 1 核上 *RKNN_NPU_CORE_2* 运行在 NPU 2 核上 *RKNN_NPU_CORE_0_1* 同时运行在 NPU 0、NPU 1 核上 *RKNN_NPU_CORE_0_1_2* 同时运行在 NPU 0、NPU 1、NPU 2 核上 *RKNN_NPU_CORE_ALL* 运行在所有的 NPU 核心上

示例代码如下：

```
#在 rknn_init 成功调用后，添加下面的接口
// Set NPU Core Mask
    rknn_core_mask core_mask = RKNN_NPU_CORE_0_1; //选择第 0 号和第 1 号 NPU 核心（可
                                                    根据需求修改）
    ret = rknn_set_core_mask(ctx, core_mask);
    if (ret != RKNN_SUCC) {
        printf("Failed to set NPU core mask! ret=%d\n", ret);
        rknn_destroy(ctx); //初始化失败时销毁上下文
        return -1;
    }
```

上述代码添加并重新编译运行之后，SH603ZA 模块的 NPU 将以双核模式运行。由于模块 NPU 默认为单核模式，若需重新改回单核模式，只需注释掉此段代码或者指定相应 NPU 核心重新编译运行即可。

5.6. 设置高性能模式进行推理

使用如下命令设置为高性能模式：

```
echo performance | tee $(find /sys/ -name *governor) /dev/null || true
```

该命令会将所有可找到的系统调频策略文件（*governor*）设置为 *performance* 模式，强制硬件以最高性能运行，忽略节能策略。

6 附录 术语缩写

表 3：术语缩写

缩写	英文全称	中文全称
ADB	Android Debug Bridge	安卓调试桥
AI	Artificial Intelligence	人工智能
API	Application Programming Interface	应用程序编程接口
BGR	Blue Green Red	蓝色绿色红色
CPU	Central Processing Unit	中央处理器
EVB	Evaluation Board	评估板
ID	Identifier	标识符
NPU	Neural-network Processing Unit	神经网络处理器
ONNX	Open Neural Network Exchange	开放神经网络交换格式
PC	Personal Computer	个人电脑
PCBA	Printed Circuit Board Assembly	印刷电路板组件
RKNN	Rockchip Neural Network	瑞芯微神经网络
SDK	Software Development Kit	软件开发工具包
SoC	System on Chip	片上系统
USB	Universal Serial Bus	通用串行总线